arXiv:1110.3959v1 [cs.ET] 18 Oct 2011

Parallel Algorithms for DNA Probe Placement on Small Oligonucleotide Arrays

Dragoş Trincă and Sanguthevar Rajasekaran
Department of Computer Science and Engineering
University of Connecticut
Storrs, CT 06269, USA
{dtrinca,rajasek}@engr.uconn.edu

October 19, 2011

Abstract

Oligonucleotide arrays are used in a wide range of genomic analyses, such as gene expression profiling, comparative genomic hybridization, chromatin immunoprecipitation, SNP detection, etc. During fabrication, the sites of an oligonucleotide array are selectively exposed to light in order to activate oligonucleotides for further synthesis. Optical effects can cause unwanted illumination at masked sites that are adjacent to the sites intentionally exposed to light. This results in synthesis of unforeseen sequences in masked sites and compromises interpretation of experimental data. To reduce such uncertainty, one can exploit freedom in how probes are assigned to array sites. The *border length minimization problem* (BLMP) seeks a placement of probes that minimizes the sum of border lengths in all masks. In this paper, we propose two parallel algorithms for the BLMP. The proposed parallel algorithms have the local-search paradigm at their core, and are especially developed for the BLMP. The results reported show that, for small microarrays with at most 1156 probes, the proposed parallel algorithms perform better than the best previous algorithms.

1 Introduction

Oligonucleotide arrays, such as those produced by Affymetrix [1], are used in a wide range of genomic analyses. As discussed in [2, 3], during very large-scale immobilized polymer synthesis (VLSIPS) the sites of a DNA probe array are selectively exposed to light in order to activate oligonucleotides for further synthesis. The selective exposure is achieved by a sequence of masks, with each mask consisting of nontransparent and transparent regions corresponding to the masked and exposed array sites. Optical effects (diffraction, reflections, etc.) can cause unwanted illumination at masked sites that are adjacent to the sites intentionally exposed to light - i.e., at the border sites of transparent regions in the mask. This results in synthesis of unforeseen sequences in masked sites and compromises interpretation of experimental data. To reduce such uncertainty, one can exploit freedom in how probes are assigned to array sites. The *border length minimization problem* (BLMP) seeks a placement of probes that minimizes the sum of border lengths in all masks. In this paper, we consider the synchronous version of the BLMP, which can be formulated as follows.

BLMP: Given a set SP consisting of dim^2 DNA sequences (called *probes*) of the same length, place the probes in SP on a $dim \times dim$ microarray in such a way that the sum of the Hamming distances between every two neighbors on the microarray is minimized. (Two probes on the microarray are said to be *neighbors* if: (1) they are adjacent and (2) they are on the same row or column of the microarray. Thus, for a $dim \times dim$ microarray, there are $dim * (dim - 1) * 2$ distinct pairs of neighbors.)

The BLMP is important not only for arrays fabricated by Affymetrix, but also for any other in-situ synthesis scheme, such as the highly-efficient micromirror arrays [4, 5, 6], or the membrane-based microarrays [12].

Previous work on the BLMP consists of the following heuristics: the TSP+1-Threading algorithm proposed in [7], the epitaxial algorithm proposed in [3], the row-epitaxial algorithm proposed in [8], and the recursive partitioning algorithm proposed in [9]. We detail these heuristics as follows.

1. The TSP+1-Threading heuristic proposed in [7] consists of two steps: (1) arrange the probes in a TSP tour by applying an approximation algorithm for the TSP, and then (2) place the sequence obtained in the first step on the microarray using the 1-Threading model, as described in [7].

2. The epitaxial heuristic proposed in [3] works as follows. Initially, a random probe in SP is placed on a random location on the microarray, and then removed from SP. Then, as long as there is at least one probe in SP, do the following: randomly select an empty location on the microarray out of those with a maximum number of neighbors, and on that location place one of the probes in SP that minimizes the sum of the Hamming distances between that probe and the current neighbors of the location (such a probe is randomly chosen if there are multiple probes that give the same minimum). Once the probe is placed, it is removed from SP.

3. The row-epitaxial heuristic proposed in [8] is similar to the epitaxial heuristic. The main difference consists of the fact that instead of placing the probes one by one, it re-shuffles an already existing pre-optimized placement.

4. The recursive partitioning heuristic proposed in [9] partitions the set of probes into subsets of the same size, and then places the probes in each subset on the corresponding submicroarray.

In this paper, we propose two parallel algorithms for the BLMP that are shown to give better results than the previous algorithms for small microarrays with up to 1156 probes. Such algorithms are especially useful for companies like:

1. SABiosciences [12], which fabricates custom microarrays with just 440 probes;

2. Febit [6], which fabricates small microarrays with just a few thousand probes.

2 Results and Discussion

In this section, we propose several algorithms for the BLMP. The algorithms that we propose are based on the local-search paradigm [10], but are more involved, and especially designed for the BLMP.

2.1 A Local-Search-based Sequential Algorithm

In this section, we propose a local-search-based sequential algorithm for the BLMP, called LS. It is given in Fig. 1. It works as follows. It takes as input the set of probes, a time limit T, and a probability parameter pr. Initially, the probes are randomly placed on the microarray. As long as the time limit was not exceeded yet, the algorithm randomly selects two locations, say (l_1, c_1) and (l_2, c_2). If swapping the probes currently on these two locations leads to a decreasing in the cost, then they are swapped and the cost is updated. Otherwise, they are swapped only with a certain probability. When the time limit is exceeded, the best microarray configuration found during the algorithm is returned.

2.2 A Local-Search-based Parallel Algorithm

The local-search-based idea can be easily parallelized, as shown in Fig. 2. The parallel variant is called LS-Par. In the first step, processor P_1 places the probes randomly on its microarray, and then sends its microarray configuration to all the other processors. So, when reaching line 3, each processor has the same configuration. At the end of each iteration through the WHILE loop that starts at line 11, the processors synchronize with each other. Let P_{source} be randomly selected out of those processors with a minimum $COST$. All the other processors update their variables with the corresponding variables from P_{source}. So, in conclusion, all the processors enter and exit every iteration through the WHILE loop with the same microarray configuration. The final result is returned by one of the processors, say P_1.

2.3 ALG1

Let $P_1, P_2, \ldots, P_k$ be the available processors. In this section, we propose a parallel algorithm for the BLMP, called ALG1, which is given in Fig. 4. The code shown in Fig. 4 is executed by each of the k processors separately. At each step, the processors synchronize with each other and (possibly) exchange some data. ALG1 incorporates the local-search idea we have seen in LS and LS-Par, but is more complicated, and especially developed for the BLMP.

The details are as follows. Each of the processors takes as input the same parameters, namely: a set of probes SP, a time limit T, and positive integers *probelength*, *MaxTrials1*, *MaxTrials2*, *MaxCost1*, *MaxCost2*, *winlength*1, and *winlength2*. The goal is to find a placement of the probes in SP on the microarray as close as possible to the optimal.

The algorithm proceeds as follows. First, processor P_1 places the probes in SP on the microarray, and then sends its microarray configuration to all the other processors. So, initially, all the processors have the same placement on the microarray. Then, each of the processors copies its microarray to *bestmicroarray*, where *bestmicroarray* is the microarray that keeps the best configuration found during the algorithm. So, initially, each processor has the same *bestmicroarray*. In lines 6-9, each of the processors computes the $COST$ of the initial microarray configuration, so each of the processors has the same $COST$. Also, $bestCOST$ is the cost corresponding to the *bestmicroarray* configuration. In *average*, each of the processors keeps the current average Hamming distance between any two neighbors on the current microarray configuration. (Note that $dim * (dim - 1) * 2$ is the total number of pairs of neighbors on a microarray with dim^2 locations.)

The basic step of ALG1 starts at line 15 and ends at line 66. It is repeated until the time taken by the algorithm exceeds T. Each of the processors repeats this basic step the same number of times. If one of the processors exits the WHILE loop that starts at line 14, then it will notify the other processors, so that the other processors will not wait for the synchronizations that start at lines 36 and 47.

During each basic step, each of the processors tries to find a pair of locations on the current microarray, say (l_1, c_1) and (l_2, c_2), with the property that swapping the probes that are currently on those locations, namely $microarray[l_1, c_1]$ and $microarray[l_2, c_2]$, will lead to a decreasing in the cost (or to a cost equal to the current cost). The pair of locations that are examined during each basic step are randomly generated, and depend on the current processor, since each processor has its own random number generator. The IF statement that starts at line 32 tries to see if swapping the probes on the chosen locations leads to an increasing in the cost (or to a cost equal to the current cost). If yes, then the probes are swapped, the OK variable is set to 1 (meaning that the current basic step is finished), and the $COST$ variable is updated accordingly. After the IF statement that starts at line 32, the processors synchronize with each other. If at least one of them, say P_{source}, has $OK = 1$, then that means that at least one of them has succeeded in finding a pair of locations that leads to a decreasing in the $COST$ (or to a cost equal to the current $COST$). If so, then all the processors (including those that have $OK = 1$) update their microarray configuration with the microarray from P_{source}. In such a case, all the processors will have $OK = 1$ after the synchronization that starts at line 36, and thus, all of them will exit the WHILE loop that starts at line 17.

If none of the processors has $OK = 1$, then all the processors will have $OK = 0$ after the synchronization that starts at line 36, and thus, all of them will enter the IF statement that starts at line 40, and then synchronize with each other at line 47. If, when reaching line 47, at least one of them, say P_{source}, has $OK = 1$, then that means that all the other processors will update their *microarray* and $COST$ with the corresponding variables from P_{source}, and then set their OK variable to 1. (At line 42, *average* is multiplied by 8 since the two chosen locations have at most 8 neighbors in total.)

So, in conclusion, all the processors exit the WHILE loop that starts at line 17 with the same nrt, meaning that each of the processors has tried the same number of pairs of locations before setting its OK variable to 1 (or reaching the *MaxTrials2* limit at line 51). Having this said, it is clear that *MaxTrials2* is meant to help the processors exit the WHILE loop that starts at line 17, and not let them run indefinitely in case that $OK = 0$ at all the processors after each processor has tried at least *MaxTrials2* pairs of locations. Also, note that all the processors enter and exit the WHILE loop that starts at line 17 with the same microarray configuration. This also implies that all of them will eventually exit the WHILE loop that starts at line 14 with the same microarray configuration and the same *bestmicroarray*.

In lines 53-66, each processor updates its corresponding parameters. The *average* variable is updated according to the new $COST$. In case that $COST < bestCOST$, then $bestCOST$ and *bestmicroarray* are updated accordingly at each processor. If $myub = 0$ and the WHILE loop that starts at line 17 was executed at least *winlength1* times since the last update of *myub*, then *myub* is set to *MaxCost1*. Otherwise, if $myub = MaxCost1$ and the WHILE loop that starts at line 17 was executed at least *winlength2* times since the last update of *myub*, then *myub* is set to 0. So, in other words, when $myub = MaxCost1$, the processors are allowed to shuffle more probes on the microarray (at line 42). This helps the algorithm to converge much faster to an approximate solution. The parameter *MaxCost2* at line 43 allows the processors to swap the probes as long as the overall cost of the resulting microarray configuration is under a certain threshold.

Example 1 *To see how ALG1 works, we give an example with three processors P_1, P_2, P_3, for a microarray of size 4×4. The input parameters are as follows:*

- *SP has 16 probes, each of length $probelength = 5$;*

- *T is 10 seconds;*

- *$dim = 4$, meaning that the microarray is of size 4×4;*

- *MaxTrials1* $= 2$;
- *MaxTrials2* $= 1000$;
- *MaxCost1* $= 10$;
- *MaxCost2* $= 10$;
- *winlength1* $= 70$;
- *winlength2* $= 20$.

The probes in SP are as given in Table 1. The algorithm works as follows.

INIT: *Processor P_1 randomly places the probes on its microarray. Without loss of generality, suppose that P_1 places the probes as shown in Fig. 3. Processor P_1 sends its microarray configuration to P_2 and P_3. So, initially, all the processors have the same microarray configuration. In such a case, it can be seen that the initial cost is 85 (so, bestCOST is 85 as well). This implies that initially, average $= 3.54$.*

Step 1: *At the beginning of this step, OK $= 0$ and nrt $= 0$. Also, sa $= 0$, myub $= 0$, and average $= 3.54$. COST and bestCOST are both 85.*

Substep 1: *Suppose that processor P_1 randomly selects locations $(1,1)$ and $(4,3)$. For these locations, localcost $= 16$, whereas newlocalcost $= 17$. Processor P_2 randomly selects locations $(3,3)$ and $(1,1)$. For these locations, localcost $= 22$ and newlocalcost $= 20$. Processor P_3 randomly selects locations $(4,2)$ and $(1,4)$. For these locations, localcost $= 14$ and newlocalcost $= 15$. So, only processor P_2 enters the IF statement that starts at line 32. So, P_2 will reach the synchronization that starts at line 36 with OK $= 1$ and COST $= 83$, and thus it will be the source processor. So, all the other processors exit the WHILE loop that starts at line 17.*

Update: *At the beginning of this phase, all the processors have COST $= 83$. The average variable becomes 3.45, bestCOST and bestmicroarray are updated accordingly, sa becomes 1, and myub remains 0.*

Step 2: *At the beginning of this step, OK $= 0$ and nrt $= 0$. Also, sa $= 1$, myub $= 0$, and average $= 3.45$. COST and bestCOST are both 83.*

Substep 1: *Suppose that P_1 selects $(1,2)$ and $(4,1)$, P_2 selects $(1,2)$ and $(4,2)$, and P_3 selects $(3,1)$ and $(4,3)$. For P_1, localcost $= 18$ and newlocalcost $= 18$. For P_2, localcost $= 22$ and newlocalcost $= 18$. For P_3, localcost $= 19$, whereas newlocalcost $= 24$. So, only P_1 and P_2 enter the IF statement that starts at line 32. Suppose that P_1 is randomly chosen to be the source processor. So, all the processors exit the WHILE loop that starts at line 17 (with COST $= 83$).*

Update: *At this point, all the processors have COST $= 83$. All the other variables remain unchanged, except for sa, which becomes 2.*

Step 3: *At the beginning of this step, OK $= 0$ and nrt $= 0$. Also, sa $= 2$, myub $= 0$, and average $= 3.45$. COST and bestCOST are both 83.*

Substep 1: *Suppose that P_1 selects $(1,4)$ and $(3,3)$, P_2 selects $(4,2)$ and $(1,2)$, and P_3 selects $(1,2)$ and $(2,4)$. For P_1, localcost $= 19$ and newlocalcost $= 24$. For P_2, localcost $= 20$ and newlocalcost $= 20$. For P_3, localcost $= 20$, whereas newlocalcost $= 23$. So, only P_2 enters the IF statement that starts at line 32, and thus P_2 is the source processor. So, all the processors exit the WHILE loop that starts at line 17 (with COST $= 83$).*

Update: *At this point, all the processors have COST $= 83$. All the other variables remain unchanged, except for sa, which becomes 3.*

Step 4: *At the beginning of this step, OK $= 0$ and nrt $= 0$. Also, sa $= 3$, myub $= 0$, and average $= 3.45$. COST and bestCOST are both 83.*

Substep 1: *Suppose that P_1 selects $(1,1)$ and $(2,2)$, P_2 selects $(2,3)$ and $(4,4)$, and P_3 selects $(1,4)$ and $(4,4)$. For P_1, localcost $= 20$ and newlocalcost $= 22$. For P_2, localcost $= 21$ and newlocalcost $= 24$. For P_3, localcost $= 12$, whereas newlocalcost $= 14$. So, none of the processors enters the IF statement that starts at line 32. Since nrt $<$ MaxTrials1 at all the processors, none of the processors swaps the probes at its selected locations. So, all the processors remain with OK $= 0$ at the end of this substep.*

Substep 2: *Suppose that* P_1 *selects* $(1,4)$ *and* $(2,4)$*,* P_2 *selects* $(1,4)$ *and* $(2,2)$*, and* P_3 *selects* $(4,1)$ *and* $(2,1)$*. For* P_1*, localcost* $= 16$ *and newlocalcost* $= 17$*. For* P_2*, localcost* $= 18$ *and newlocalcost* $= 22$*. For* P_3*, localcost* $= 18$*, whereas newlocalcost* $= 19$*. So, none of the processors enters the IF statement that starts at line 32. Since nrt* $=$ *MaxTrials1 and the other conditions (at lines 42-43) are satisfied at all the processors, each of the processors swaps the probes at its selected locations. Suppose that* P_1 *is chosen to be the source processor at the synchronization that starts at line 47. Thus, at the end of this substep, all the processors have COST* $= 84$*, and, since OK* $= 1$ *at all the processors, all of them exit the WHILE loop that starts at line 17.*

Update: *At this point, each processors has COST* $= 84$*. The average variable becomes* 3.50*, bestCOST and bestmicroarray remain unchanged (since COST just increased), sa becomes* 4*, and myub remains* 0*.*

Step 5: *Suppose that at this point, the time limit T is exceeded at all the processors. Thus, all of them exit the WHILE loop that starts at line 14. Only processor* P_1 *returns the best cost (and the corresponding bestmicroarray configuration) found during the algorithm, which is* 83*.*

2.4 ALG2

In this section we propose a variant of ALG1, called ALG2, which is given in Fig. 5. The only difference from ALG1 is that we have a new input parameter, namely *MaxCost*, which replaces the *average* variable used in ALG1. This will allow ALG2 to give better results than ALG1 for some microarray dimensions.

2.5 Results (on Randomly Generated Sets of Probes)

We have implemented the previous heuristics (TSP+1-Threading, epitaxial, row-epitaxial, recursive partitioning) and the algorithms discussed in this paper (LS, LS-Par, ALG1, and ALG2) on a SGI Altix machine with 64 processors, using MPI [11]. Since the previous four heuristics (TSP+1-Threading, epitaxial, row-epitaxial, recursive partitioning) and the LS algorithm are sequential algorithms, we have run them using only one processor out of 64 available. For LS-Par, ALG1, and ALG2, we have used all 64 processors available in order to help them to converge faster to an approximate solution. For small microarrays, each of the four previous heuristics (TSP+1-Threading, epitaxial, row-epitaxial, recursive partitioning) takes just a few seconds. We can definitely use up to 64 processors in order to reduce the time taken by each of them even further, but this does not help in reducing the cost. Indeed, the four previous heuristics (TSP+1-Threading, epitaxial, row-epitaxial, recursive partitioning), unlike LS, LS-Par, ALG1, and ALG2, are algorithms with a finite number of steps. For small microarrays of size at most 34 × 34, the epitaxial algorithm gives better results than TSP+1-Threading, row-epitaxial, and recursive partitioning. Thus, we compare the epitaxial algorithm against LS, LS-Par, ALG1, and ALG2.

We have run LS with different probabilities, and collected results after 2, 4, 6, 8, and 10 minutes. For LS-Par, we have used all 64 processors available, and collected results after 2, 4, 6, 8, and 10 minutes. For all microarray dimensions considered, LS and LS-Par perform worse than the epitaxial algorithm. We have also implemented ALG1 and ALG2 using all 64 processors available, and collected results after 2, 4, 6, 8, and 10 minutes. The parameters used in order to get to the results shown in Tables 3 and 4 are as follows: *probelength* $= 25$, *MaxTrials1* $= 20$, *MaxTrials2* $= 40000$, *MaxCost* $= 160$, *MaxCost1* $= 10$, *MaxCost2* $= 10$, *winlength1* $= 1120$, *winlength2* $= 320$. For microarrays of size at most 32×32, ALG1 gives better results than the epitaxial algorithm. For microarrays of size 33×33 or more, ALG1 gives worse results than the epitaxial algorithm. For microarrays of size at most 34×34, ALG2 gives better results than the epitaxial algorithm. For microarrays of size 35×35 or more, ALG2 gives worse results than the epitaxial algorithm. We can also remark that ALG2 performs better than ALG1. This suggests that the *MaxCost* input parameter in ALG2 is more helpful than the *average* variable in ALG1.

References

[1] **Affymetrix, Inc.:** http://www.affymetrix.com.

[2] Fodor S, Read JL, Pirrung MC, Stryer L, Tsai LA, Solas D: **Light-Directed, Spatially Addressable Parallel Chemical Synthesis**. *Science* 1991, **251**:767–773.

[3] Kahng AB, Mandoiu II, Pevzner P, Reda S, Zelikovsky A: **Border Length Minimization in DNA Array Design**. In *Proceedings of the 2nd Workshop on Algorithms in Bioinformatics*, Springer LNCS 2002:435–448.

[4] Singh-Gasson S, Green RD, Yue Y, Nelson C, Blattner F, Sussman MR, Cerrina F: **Maskless Fabrication of Light-Directed Oligonucleotide Microarrays using a Digital Micromirror Array**. *Nature Biotechnology* 1999, **17**:974–978.

[5] **Invitrogen, Inc.:** http://www.invitrogen.com.

[6] **Febit, Inc.:** http://www.febit.com.

[7] Hannenhalli S, Hubbell E, Lipshutz R, Pevzner PA: **Combinatorial Algorithms for Design of DNA Arrays**. In *Chip Technology.* Edited by Hoheisel J, Springer 2002.

[8] Kahng AB, Mandoiu II, Pevzner P, Reda S, Zelikovsky A: **Engineering a Scalable Placement Heuristic for DNA Probe Arrays**. In *Proceedings of the 7th Annual International Conference on Research in Computational Molecular Biology*, ACM 2003:148–156.

[9] Kahng AB, Mandoiu II, Reda S, Xu X, Zelikovsky A: **Evaluation of Placement Techniques for DNA Probe Array Layout**. In *Proceedings of the IEEE-ACM International Conference on Computer-Aided Design*, IEEE Press 2003:262–269.

[10] Papadimitriou CH, Steiglitz K: *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications 1998.

[11] **The MPI standard:** http://www-unix.mcs.anl.gov/mpi.

[12] **SABiosciences, Inc.:** http://www.sabiosciences.com/custom.php.

Table 1: The 16 probes in SP

p_1 = CGATT	p_9 = ATACG
p_2 = GGGCC	p_{10} = CCCTC
p_3 = ATCGA	p_{11} = GGAGA
p_4 = ATGTC	p_{12} = AGCCG
p_5 = TTAGT	p_{13} = AGACA
p_6 = ACCAG	p_{14} = ACCTA
p_7 = CCCGA	p_{15} = GAATC
p_8 = AATTC	p_{16} = GATTT

```
Input: $SP$, $T$, $dim$, $probelength$, and an input probability $pr$
Output: a microarray configuration as close as possible to the optimal

• Place the probes in $SP$ randomly on the $microarray$.
• FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim\}$ DO
        • $bestmicroarray[i, j] \leftarrow microarray[i, j]$;
• $COST \leftarrow 0$;
• FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim - 1\}$ DO
        • $COST \leftarrow COST + HammingDistance(microarray[i, j], microarray[i, j + 1])$;
• FOR all $j \in \{1, \ldots, dim\}$ and all $i \in \{1, \ldots, dim - 1\}$ DO
        • $COST \leftarrow COST + HammingDistance(microarray[i, j], microarray[i + 1, j])$;
• $bestCOST \leftarrow COST$;
• WHILE (the time taken by the algorithm is $\leq T$) DO
        • Choose two random locations on the $microarray$, say $(l_1, c_1)$ and $(l_2, c_2)$. (The random locations that are chosen depend on the current processor, since each of the processors has its own random number generator.)
        • Let $localcost1$ be the sum of the Hamming distances between the probe $microarray[l_1, c_1]$ and the probes that are currently neighbors to $(l_1, c_1)$.
        • Let $localcost2$ be the sum of the Hamming distances between the probe $microarray[l_2, c_2]$ and the probes that are currently neighbors to $(l_2, c_2)$.
        • Let $newlocalcost1$ be the sum of the Hamming distances between the probe $microarray[l_2, c_2]$ and the probes that are currently neighbors to $(l_1, c_1)$.
        • Let $newlocalcost2$ be the sum of the Hamming distances between the probe $microarray[l_1, c_1]$ and the probes that are currently neighbors to $(l_2, c_2)$.
        • $localcost \leftarrow localcost1 + localcost2$;
        • $newlocalcost \leftarrow newlocalcost1 + newlocalcost2$;
        • IF ($newlocalcost < localcost$) THEN
                • Swap the probes at locations $(l_1, c_1)$ and $(l_2, c_2)$;
                • $COST \leftarrow COST - (localcost - newlocalcost)$;
                • $bestCOST \leftarrow COST$;
                • FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim\}$ DO
                        • $bestmicroarray[i, j] \leftarrow microarray[i, j]$;
        ELSE
                • Randomly generate a number $N$ in the interval $[0, 1]$.
                • IF $N \leq pr$ THEN
                        • Swap the probes at locations $(l_1, c_1)$ and $(l_2, c_2)$;
                        • $COST \leftarrow COST + (newlocalcost - localcost)$;
• return $bestmicroarray$;
```

Figure 1: LS: a local-search-based sequential algorithm for the BLMP

```
Input: $SP$, $T$, $dim$, $probelength$, and an input probability $pr$
Output: a microarray configuration as close as possible to the optimal
------------------------------------------------------------------------
• Processor $P_1$ places the probes in $SP$ randomly on its $microarray$ and then sends its microarray configuration
  to all the other processors.
• FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim\}$ DO
        • $bestmicroarray[i, j] \leftarrow microarray[i, j]$;
• $COST \leftarrow 0$;
• FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim - 1\}$ DO
        • $COST \leftarrow COST + HammingDistance(microarray[i, j], microarray[i, j + 1])$;
• FOR all $j \in \{1, \ldots, dim\}$ and all $i \in \{1, \ldots, dim - 1\}$ DO
        • $COST \leftarrow COST + HammingDistance(microarray[i, j], microarray[i + 1, j])$;
• $bestCOST \leftarrow COST$;
• WHILE (the time taken by the algorithm is $\leq T$) DO
        • Choose two random locations on the $microarray$, say $(l_1, c_1)$ and $(l_2, c_2)$. (The random locations that
          are chosen depend on the current processor, since each of the processors has its own random number
          generator.)
        • Let $localcost1$ be the sum of the Hamming distances between the probe $microarray[l_1, c_1]$ and the
          probes that are currently neighbors to $(l_1, c_1)$.
        • Let $localcost2$ be the sum of the Hamming distances between the probe $microarray[l_2, c_2]$ and the
          probes that are currently neighbors to $(l_2, c_2)$.
        • Let $newlocalcost1$ be the sum of the Hamming distances between the probe $microarray[l_2, c_2]$ and
          the probes that are currently neighbors to $(l_1, c_1)$.
        • Let $newlocalcost2$ be the sum of the Hamming distances between the probe $microarray[l_1, c_1]$ and
          the probes that are currently neighbors to $(l_2, c_2)$.
        • $localcost \leftarrow localcost1 + localcost2$;
        • $newlocalcost \leftarrow newlocalcost1 + newlocalcost2$;
        • IF $(newlocalcost < localcost)$ THEN
                • Swap the probes at locations $(l_1, c_1)$ and $(l_2, c_2)$;
                • $COST \leftarrow COST - (localcost - newlocalcost)$;
                • $bestCOST \leftarrow COST$;
                • FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim\}$ DO
                        • $bestmicroarray[i, j] \leftarrow microarray[i, j]$;
          ELSE
                • Randomly generate a number $N$ in the interval $[0, 1]$.
                • IF $N \leq pr$ THEN
                        • Swap the probes at locations $(l_1, c_1)$ and $(l_2, c_2)$;
                        • $COST \leftarrow COST + (newlocalcost - localcost)$;
        • Processors synchronize with each other. Let $P_{source}$ be randomly selected out of those processors that
          have a minimum $COST$. All the other processors update their $microarray$, $COST$, $bestmicroarray$,
          $bestCOST$ with the corresponding variables from $P_{source}$.
• Processor $P_1$ returns $bestmicroarray$;
```

Figure 2: LS-Par: a local-search-based parallel algorithm for the BLMP (The pseudocode shown here is executed by each of the processors; only processor P_1 returns the final result.)

p_1	p_2	p_3	p_4
p_5	p_6	p_7	p_8
p_9	p_{10}	p_{11}	p_{12}
p_{13}	p_{14}	p_{15}	p_{16}

Figure 3: The initial placement (at all the processors) of the probes on the microarray

Table 2: The Hamming distances between every two probes in SP

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}
p_1		4	5	4	3	5	4	4	4	3	3	4	3	4	3	3
p_2			5	3	5	5	5	4	4	4	3	3	3	5	3	4
p_3				3	3	3	2	4	3	4	3	3	3	2	5	5
p_4					4	4	5	2	3	3	5	4	4	3	3	4
p_5						5	4	5	3	5	3	5	4	5	4	4
p_6							3	4	3	3	5	2	4	2	5	5
p_7								5	5	2	3	4	4	2	5	5
p_8									4	3	5	4	4	3	2	2
p_9										5	4	2	2	4	4	5
p_{10}											5	4	5	2	3	4
p_{11}												4	2	4	3	4
p_{12}													2	3	5	5
p_{13}														3	4	5
p_{14}															4	4
p_{15}																2
p_{16}																

```
Input: $SP$, $T$, $dim$, $probelength$, $MaxTrials1$, $MaxTrials2$, $MaxCost1$, $MaxCost2$, $winlength1$, $winlength2$
Output: a microarray configuration as close as possible to the optimal

 1: • Processor $P_1$ places all the probes in $SP$ randomly on its $microarray$, and then sends its $microarray$ to all
 2:   the other processors;
 3: • FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim\}$ DO
 4:         • $bestmicroarray[i, j] \leftarrow microarray[i, j]$;
 5: • $COST \leftarrow 0$;
 6: • FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim - 1\}$ DO
 7:         • $COST \leftarrow COST + HammingDistance(microarray[i, j], microarray[i, j + 1])$;
 8: • FOR all $j \in \{1, \ldots, dim\}$ and all $i \in \{1, \ldots, dim - 1\}$ DO
 9:         • $COST \leftarrow COST + HammingDistance(microarray[i, j], microarray[i + 1, j])$;
10: • $average \leftarrow [COST/(dim * (dim - 1) * 2)]$;
11: • $bestCOST \leftarrow COST$;
12: • $sa \leftarrow 0$;
13: • $myub \leftarrow 0$;
14: • WHILE (the time taken by the algorithm is $\leq T$) DO
15:         • $OK \leftarrow 0$;
16:         • $nrt \leftarrow 0$;
17:         • WHILE $(OK = 0)$ DO
18:                 • $nrt \leftarrow nrt + 1$;
19:                 • Choose two random locations on the $microarray$, say $(l_1, c_1)$ and $(l_2, c_2)$. (The random
20:                   locations that are chosen depend on the current processor, since each processor has its
21:                   own random number generator.)
22:                 • Let $localcost1$ be the sum of the Hamming distances between the probe $microarray[l_1, c_1]$
23:                   and the probes that are currently neighbors to $(l_1, c_1)$.
24:                 • Let $localcost2$ be the sum of the Hamming distances between the probe $microarray[l_2, c_2]$
25:                   and the probes that are currently neighbors to $(l_2, c_2)$.
26:                 • Let $newlocalcost1$ be the sum of the Hamming distances between the probe $microarray[l_2, c_2]$
27:                   and the probes that are currently neighbors to $(l_1, c_1)$.
28:                 • Let $newlocalcost2$ be the sum of the Hamming distances between the probe $microarray[l_1, c_1]$
29:                   that are currently neighbors to $(l_2, c_2)$.
30:                 • $localcost \leftarrow localcost1 + localcost2$;
31:                 • $newlocalcost \leftarrow newlocalcost1 + newlocalcost2$;
32:                 • IF $(newlocalcost \leq localcost)$ THEN
33:                         • $OK \leftarrow 1$;
34:                         • Swap the probes at locations $(l_1, c_1)$ and $(l_2, c_2)$;
35:                         • $COST \leftarrow COST - (localcost - newlocalcost)$;
36:                 • Processors synchronize with each other. If at least one of them has $OK = 1$, then let $P_{source}$
37:                   be one of those processors with $OK = 1$, randomly chosen. All the other processors update
38:                   their $microarray$ and $COST$ with the corresponding variables from $P_{source}$, and then set their
39:                   $OK$ their variable to 1.
40:                 • IF $(OK = 0)$ THEN
41:                         • IF $(nrt \geq MaxTrials1)$ THEN
42:                                 • IF $(newlocalcost \leq \lfloor 8 * average \rfloor + myub)$ THEN
43:                                         • IF $(COST + newlocalcost - localcost \leq bestCOST + MaxCost2)$ THEN
44:                                                 • $OK \leftarrow 1$;
45:                                                 • Swap the probes at locations $(l_1, c_1)$ and $(l_2, c_2)$;
46:                                                 • $COST \leftarrow COST + (newlocalcost - localcost)$;
47:                         • Processors synchronize with each other. If at least one of them has $OK = 1$, then let
48:                           $P_{source}$ be one of those processors with $OK = 1$, randomly chosen. All the other
49:                           processors update their $microarray$ and $COST$ with the corresponding variables from
50:                           $P_{source}$, and then set their $OK$ variable to 1.
51:                 • IF $(OK = 0)$ and $(nrt \geq MaxTrials2)$ THEN
52:                         • BREAK the WHILE that starts at line 17;
53:         • $average \leftarrow [COST/(dim * (dim - 1) * 2)]$;
54:         • IF $(COST < bestCOST)$ THEN
55:                 • $bestCOST \leftarrow COST$;
56:                 • FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim\}$ DO
57:                         • $bestmicroarray[i, j] \leftarrow microarray[i, j]$;
58:         • $sa \leftarrow sa + 1$;
59:         • IF $(myub = 0)$ THEN
60:                 • IF $(sa = winlength1)$ THEN
61:                         • $sa \leftarrow 0$;
62:                         • $myub \leftarrow MaxCost1$;
63:           ELSE
64:                 • IF $(sa = winlength2)$ THEN
65:                         • $sa \leftarrow 0$;
66:                         • $myub \leftarrow 0$;
67: • Processor $P_1$ returns $bestmicroarray$;
```

Figure 4: ALG1 (the pseudocode shown here is executed by each of the processors involved in the algorithm; only processor P_1 returns the final result.)

```
Input: $SP$, $T$, $dim$, $probelength$, $MaxTrials1$, $MaxTrials2$, $MaxCost$, $MaxCost1$, $MaxCost2$, $winlength1$, $winlength2$
Output: a microarray configuration as close as possible to the optimal

 1: • Processor $P_1$ places all the probes in $SP$ randomly on its $microarray$, and then sends its $microarray$ to all
 2:   the other processors;
 3: • FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim\}$ DO
 4:         • $bestmicroarray[i, j] \leftarrow microarray[i, j]$;
 5: • $COST \leftarrow 0$;
 6: • FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim - 1\}$ DO
 7:         • $COST \leftarrow COST + HammingDistance(microarray[i, j], microarray[i, j + 1])$;
 8: • FOR all $j \in \{1, \ldots, dim\}$ and all $i \in \{1, \ldots, dim - 1\}$ DO
 9:         • $COST \leftarrow COST + HammingDistance(microarray[i, j], microarray[i + 1, j])$;
10: • $bestCOST \leftarrow COST$;
11: • $sa \leftarrow 0$;
12: • $myub \leftarrow 0$;
13: • WHILE (the time taken by the algorithm is $\leq T$) DO
14:         • $OK \leftarrow 0$;
15:         • $nrt \leftarrow 0$;
16:         • WHILE ($OK = 0$) DO
17:                 • $nrt \leftarrow nrt + 1$;
18:                 • Choose two random locations on the $microarray$, say $(l_1, c_1)$ and $(l_2, c_2)$. (The random
19:                   locations that are chosen depend on the current processor, since each processor has its
20:                   own random number generator.)
21:                 • Let $localcost1$ be the sum of the Hamming distances between the probe $microarray[l_1, c_1]$
22:                   and the probes that are currently neighbors to $(l_1, c_1)$.
23:                 • Let $localcost2$ be the sum of the Hamming distances between the probe $microarray[l_2, c_2]$
24:                   and the probes that are currently neighbors to $(l_2, c_2)$.
25:                 • Let $newlocalcost1$ be the sum of the Hamming distances between the probe $microarray[l_2, c_2]$
26:                   and the probes that are currently neighbors to $(l_1, c_1)$.
27:                 • Let $newlocalcost2$ be the sum of the Hamming distances between the probe $microarray[l_1, c_1]$
28:                   that are currently neighbors to $(l_2, c_2)$.
29:                 • $localcost \leftarrow localcost1 + localcost2$;
30:                 • $newlocalcost \leftarrow newlocalcost1 + newlocalcost2$;
31:                 • IF ($newlocalcost \leq localcost$) THEN
32:                         • $OK \leftarrow 1$;
33:                         • Swap the probes at locations $(l_1, c_1)$ and $(l_2, c_2)$;
34:                         • $COST \leftarrow COST - (localcost - newlocalcost)$;
35:                 • Processors synchronize with each other. If at least one of them has $OK = 1$, then let $P_{source}$
36:                   be one of those processors with $OK = 1$, randomly chosen. All the other processors update
37:                   their $microarray$ and $COST$ with the corresponding variables from $P_{source}$, and then set their
38:                   $OK$ their variable to 1.
39:                 • IF ($OK = 0$) THEN
40:                         • IF ($nrt \geq MaxTrials1$) THEN
41:                                 • IF ($newlocalcost \leq MaxCost + myub$) THEN
42:                                         • IF ($COST + newlocalcost - localcost \leq bestCOST + MaxCost2$) THEN
43:                                                 • $OK \leftarrow 1$;
44:                                                 • Swap the probes at locations $(l_1, c_1)$ and $(l_2, c_2)$;
45:                                                 • $COST \leftarrow COST + (newlocalcost - localcost)$;
46:                         • Processors synchronize with each other. If at least one of them has $OK = 1$, then let
47:                           $P_{source}$ be one of those processors with $OK = 1$, randomly chosen. All the other
48:                           processors update their $microarray$ and $COST$ with the corresponding variables from
49:                           $P_{source}$, and then set their $OK$ variable to 1.
50:                 • IF ($OK = 0$) and ($nrt \geq MaxTrials2$) THEN
51:                         • BREAK the WHILE that starts at line 16;
52:         • IF ($COST < bestCOST$) THEN
53:                 • $bestCOST \leftarrow COST$;
54:                 • FOR all $i \in \{1, \ldots, dim\}$ and all $j \in \{1, \ldots, dim\}$ DO
55:                         • $bestmicroarray[i, j] \leftarrow microarray[i, j]$;
56:         • $sa \leftarrow sa + 1$;
57:         • IF ($myub = 0$) THEN
58:                 • IF ($sa = winlength1$) THEN
59:                         • $sa \leftarrow 0$;
60:                         • $myub \leftarrow MaxCost1$;
61:           ELSE
62:                 • IF ($sa = winlength2$) THEN
63:                         • $sa \leftarrow 0$;
64:                         • $myub \leftarrow 0$;
65: • Processor $P_1$ returns $bestmicroarray$;
```

Figure 5: ALG2 (the pseudocode shown here is executed by each of the processors involved in the algorithm; only processor P_1 returns the final result.)

Table 3: Comparisons between ALG1 (with 64 processors) and the epitaxial algorithm

		ALG1				
dim	Epitaxial	$T = 2$ min.	$T = 4$ min.	$T = 6$ min.	$T = 8$ min.	$T = 10$ min.
32	55,296	55,680	**55,238**	**55,068**	**54,942**	**54,894**
30	48,604	48,832	**48,576**	**48,428**	**48,418**	**48,392**
28	42,676	**42,298**	**42,184**	**42,088**	**42,070**	**42,040**
26	36,806	**36,656**	**36,536**	**36,528**	**36,450**	**36,382**
24	31,480	**31,074**	**31,004**	**31,004**	**31,004**	**31,004**
22	26,608	**26,208**	**26,208**	**26,208**	**26,208**	**26,208**
20	21,956	**21,698**	**21,666**	**21,666**	**21,666**	**21,666**
18	17,884	**17,588**	**17,588**	**17,588**	**17,588**	**17,588**
16	14,190	**13,916**	**13,916**	**13,916**	**13,916**	**13,916**

Table 4: Comparisons between ALG2 (with 64 processors) and the epitaxial algorithm

		ALG2				
dim	Epitaxial	$T = 2$ min.	$T = 4$ min.	$T = 6$ min.	$T = 8$ min.	$T = 10$ min.
34	62,072	63,316	62,774	62,438	62,222	**62,026**
32	55,296	56,060	55,468	**55,258**	**55,060**	**54,918**
30	48,604	48,924	**48,398**	**48,226**	**48,070**	**47,988**
28	42,676	**42,382**	**42,138**	**42,006**	**41,946**	**41,894**
26	36,806	**36,572**	**36,332**	**36,246**	**36,222**	**36,192**
24	31,480	**31,032**	**30,916**	**30,864**	**30,818**	**30,774**
22	26,608	**26,084**	**25,922**	**25,860**	**25,850**	**25,842**
20	21,956	**21,672**	**21,482**	**21,482**	**21,482**	**21,482**
18	17,884	**17,376**	**17,376**	**17,330**	**17,328**	**17,328**
16	14,190	**13,730**	**13,730**	**13,730**	**13,730**	**13,730**